

slide 7: Implementation Phase

~~What~~ → What happens in this phase:

algorithms described in design phase are converted into programming code after choosing the appropriate programming language.

→ Each programmer receives his module details as a flow chart or pseudo code, ^{and} implement his part.

→ How to choose appropriate programming language:

* each app. may have more than one appropriate programming language.

* Programmers choose any of them according to his experience.

→ How to be a good programmer?

* you must:

1) learn the theoretical background of the language.

2) practice on what you had learnt ~~into~~ to gain more skills.

3) use the experience & help from other programmers.

EX

1) Graphics can be programmed by Java, C++
~~and~~ or Visual Basic.

2) Web design: Can be programmed by PHP
or XML or HTML + JavaScript or VB ~~script~~ script.



1) Program names

- choose appropriate Program names such as names of variables, functions, classes,
- Not too small, not too long and not expensive.

Ex name of variable that hold radius of circle

* Radius - of - circle → too long

* R → too small

* RA → not expensive

* RA or Radius ✓

2) Program structure

* Program should be written in an organized form to facilitate reading, Examples:

- leave a blank line between functions ~~des~~ description.
- avoid written two or more statements in the same line (as in C++).
- Put all declaration after function header to easily locate them.
- first letter in function names, arrays, classes in capital letter.
- Add comments to complicated functions.

3) Program Control structure

* use appropriate control structure (if, while, for, switch) that will achieve the required function with less number of lines and less complexity.

Examples

```
if (c == 1) statements
else if (c == 2) statements.
else if (c == 3) statements
else statements
```

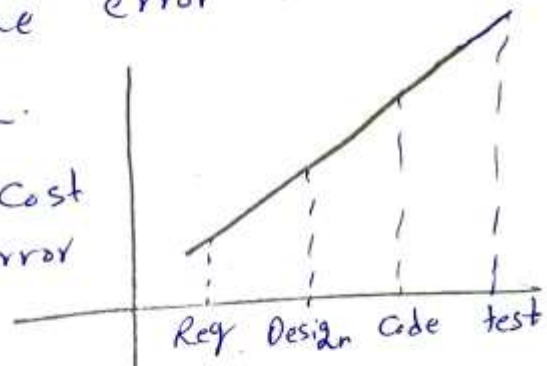
```
switch (c) : statements
Case 1: statements
Case 2: statements
Case 3: statements
Default: statements.
```

Fourth Phase: test phase

→ Test the SW is important if it is used to control sensitive devices such as:
nuclear power plant & autopilot in airplanes.

→ the earlier we find the error the less ~~the~~ the cost of its fixing.

Relative Cost
to fix error



→ It ~~shows~~ only presence of errors not absence.

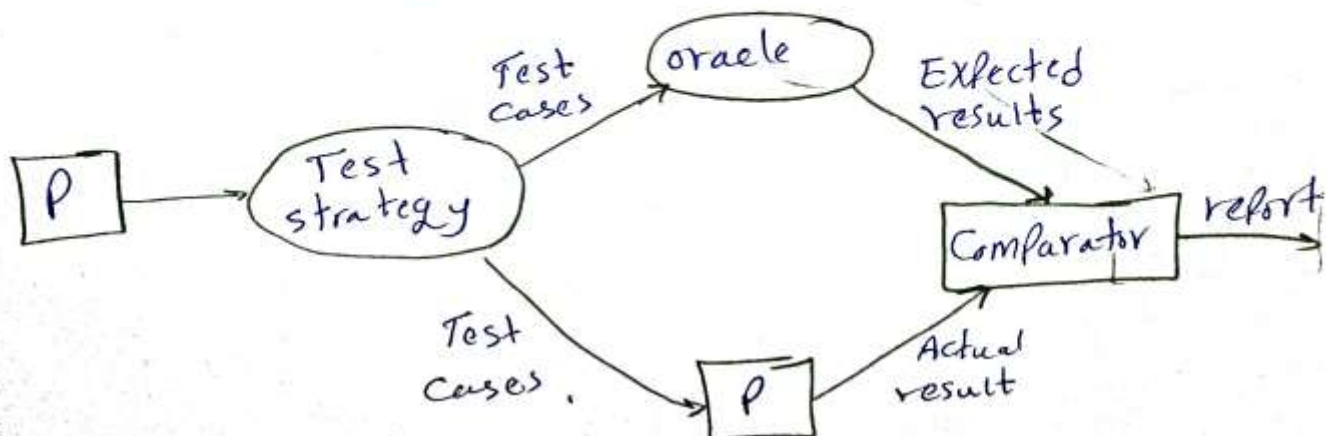
Error → human action that produces incorrect result.

Fault → The consequence of an error in SW lines.

Failure → result of fault during Program Execution.

→ at test we observe ~~error~~ failure which caused by faults, which are result of human error

Test Process steps



P: object to be tested (Program, module,)

Test strategy: decide how to test your object.

Test Cases: Conditions applied to the tested Program to test whether it performs its functions in proper way.

oracle: way to determine the expected output for the test cases.

→ test ~~cases~~ cases applied to P to produce the actual results.

Comparator → compare actual output to expected o/p.

Test adequacy criteria

→ Percentage of Program lines to be tested.

→ in critical Programms, we need to have high test adequacy criteria.

[Ex] C in $\Rightarrow x, y;$
if (x > y)
 cout << "x is greater than y";
Else y
 cout << "x is greater than ~~y~~"

test adequacy 100%
test cases {x=3, y=5}, {6, 2}

test adequacy 50%
test cases {3, 5}

←

Test techniques

* Automated test "not available"

→ uses intelligent program that receives the tested object and test cases, then it tests the object generating a report.

* Manual test

→ Human test of the project.

a) static test

→ we test program without making executable (executable is the file of the program).

~~base~~ → test made by reading the program.

a.1) Reading "no test cases are used"

→ Program is read line by line to find errors by one tester.

→ Compiler does the same task of reading test instead of human.

→ suitable for small size programs or documents.

a.2) scenario-based test (test cases are used)

→ we follow program lines imagining the scenario of program execution after entering the test cases.

a.3) walk through (no test cases are used)

→ Program is read line by line to find errors but by group of testers, one read and they walk through lines ~~any~~ finding errors and taking notes.

a.4) correctness proof (test cases are used)

- we have an expected output for test cases.
- we follow program lines after entering the test cases until we get actual result.
- we compare both results & find place of error

b. Dynamic test techniques

- we test program after making executable (exe) file of program

b.1) Coverage based test

- choose test cases that cover % of program lines.
- need flowchart to understand program structure & to determine test cases.
- program is executed no. of times with different test cases that covered required %.

b.2) Fault-based test

- This type inserts the intentional error in the Program lines, which its result is expected
- The actual result of executing the Program with the inserted errors is compared to the expected result with error.
- if they match, then there is no error except that inserted one.
- if not, there is an error(s) in the Program.

Example

Without error

Cin $\Rightarrow$ X, Z;

Z = X + Y;

Cout $\Leftarrow$ Z;

After error

Cin $\times \Rightarrow$ X, Z;

Z = X - Y;

Cout $\Leftarrow$ Z;

Test cases = {3, 5}

- expected result with error = -2
- actual result will be Garbage as Y is not initialized or set to value
- error in line 1 (Z replaced by y).

Fifth Phase: Maintenance

↳ It keeps the SW running at a proper level that satisfies the user.

⇒ That Phase Contains

- 1) repairing faults found.
- 2) Adapting to environmental changes.
- 3) Adding or modifying functionality to SW.

→ Types of maintenance activities

- | | |
|---------------|---------------|
| 1) Correcting | 2) Adapting |
| 3) Perfective | 4) Preventive |

1) Corrective Maintenance

→ Concerned with correcting any faults found by SW user after delivery.

→ These faults not discovered before delivery
Cause of:

- 1) Lost test adequacy.
- 2) Short time assigned to test phase.
- 3) Complexity of SW.

2) Adaptive maintenance

→ responsible for ~~making~~ any changes in the SW that makes it work under different environment other than the one which is was designed to work under such that:

- a) Different operating system.
- b) Different number of users.
- c) Different hardware configuration
- d) Different backgrounds or fonts

↳ variables in functions of system are changed ~~and~~ not the functions

3) Perfective maintenance

1) responsible for adding new features to SW which ~~re~~ user requires to improve the functionality of SW.

2) So, the old version of the SW is working with no faults, yet with less functions than required.

4) Preventive maintenance

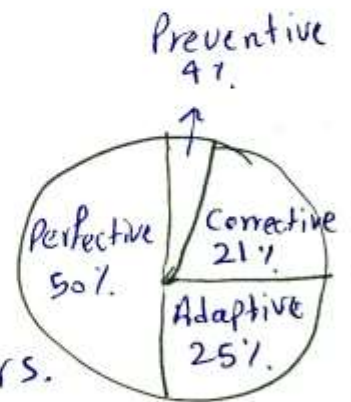
- responsible for maintaining the sw documents after making any of three previous activities.
- It is performed to keep the documents up-to-date with the working version of sw.
- It will make performing other activities.
- if it's not performed, documents will be not compatible with the working version of sw.

Maintenance activities distribution

1) Perfective is the highest cause of:

a) requirement phase was not performed in right way to capture all requirements of users.

b) demanding for change natural of the users which keeps them in continuous demand for new functions.



2) Adaptive is high cause due to

- a) changing nature of OS and HW Configurations.
- b) if flexibility of the SW itself as it did give the user the ability to make these changes himself.

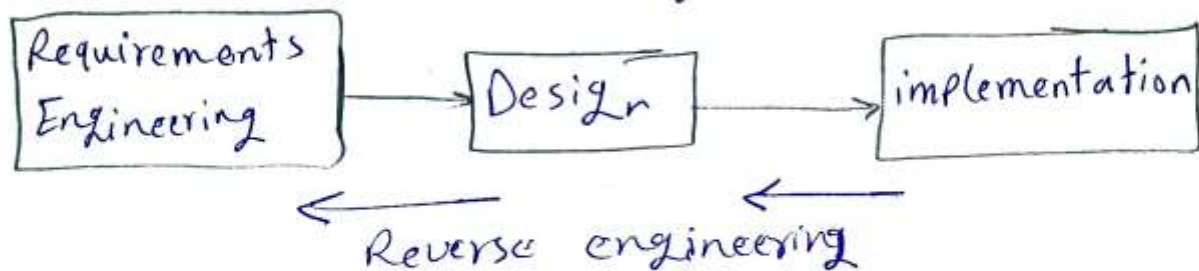
3) Corrective activity high specially in
→ complicated SW & little time given to test phase.

What are the problems that face maintenance engineer

- a) Program is not clear (name and structure)
- b) hard to understand problem domain
- c) documents may be lost or not up-to-date
- d) if design is bad, It is very hard to add new functions or to change some parameters to adapt environment changes.

Reverse SW engineering

Forward Engineering



- one of maintenance engineer possible activities
- used if SW documents are missing.
- It is the science of reproducing the SW documents from the currently working version of the SW.

SW Restructuring

- Maintenance engineer can restructure the current SW if SW structure is poor.
- This step is made without changing function of the SW.
- It only organize these function in proper way, external or internal.

Externally: by regrouping them in different menus.

Internally: changing the control structures or program names.

Maintenance Process steps

- 1) Collect informations about activities required.
- 2) organize them in a report with specifying the type of each activity, what is required, estimated cost.
- 3) This report delivered to manager who will determine priority of each step according to company's condition & client's approval.
- 4) chosen activities is scheduled to be performed
- 5) SW is tested & then delivered to users.

SW Cost estimation

* Cost of producing software =
Requirement Cost + Design cost + Test Cost
+ implementation Cost.

* traditional way to compute Cost = No. of workers

* No. of working hours * hourly salary.

→ but hourly salary is not equal in many cases cause of difference in human Experience.

Cost

$$E = 2.7 (KOLC)^2$$

KOLC = Kilo lines of code.

→ It is related to effort that done to make SW in terms of lines produced.

⇒ we get experts to produce same functions in less number of ~~code~~ lines.

⇒ we can also reuse parts from other programs to minimize new number of lines.

SW Quality

→ quality of SW that can be measured from different point of view.

→ Examples of SW quality measures.

1] Correctness:-

- Does SW perform its tasks properly?
- Are these requirements that users demanded?

2] Reliability

- Does SW performs its tasks accurately each time?

3) Flexibility

* easy to make adaption or perfection maintenance to it.

4) Portability

↳ ability to work under different environments.

5) Testability

Easy to be tested.

6) Interoperability

* Able to work with other programs at the same time.

* dp of SW can be used by other SW.

7) Reusability

* ~~extent~~ Its components can be reused in other SW building.

8) Usability

↳ The effort required to learn, operate, prepare inputs and interpret output of SW.

9) Maintainability

↳ effort required to locate and fix errors in the SW.